```
/*****************************************************************************
#   Copyright 1997 Silicon Graphics, Inc.  ALL RIGHTS RESERVED.
#
#   UNPUBLISHED -- Rights reserved under the copyright laws of the United
#   States.   Use of a copyright notice is precautionary only and does not
#   imply publication or disclosure.
#
#   THIS SOFTWARE CONTAINS CONFIDENTIAL AND PROPRIETARY INFORMATION OF
#   SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION, DISTRIBUTION, OR
#   DISCLOSURE IS STRICTLY PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN
#   PERMISSION OF SILICON GRAPHICS, INC.
#****************************************************************************/

#include <stdio.h>
#include <stdarg.h>
#include <strings.h>
#include <math.h>
#include "ri.h"
#include "ri_state.h"
#include <GL/gl.h>
#include <GL/glu.h>

typedef struct CoordSystem {
    char name[20];
    RtMatrix xform;
    RtMatrix ixform;
    struct CoordSystem *next;
} CoordSystem;

/* returns true if the current orientation is not same as the
   orientation of the current transform */
int insideout(void)
{
    if( ( (CurAttributes->orientation==RI_OUTSIDE ||
           CurAttributes->orientation==RI_LH) &&
          (CurAttributes->transorient==RI_RH) ) ||
      ( (CurAttributes->orientation==RI_INSIDE ||
           CurAttributes->orientation==RI_RH) &&
          (CurAttributes->transorient==RI_LH) ) ) {
      return RI_TRUE;
    }

    return RI_FALSE;
}

void __frontface_set(void)
{
    if( insideout() ) {
        glFrontFace(GL_CCW);
    } else {
        glFrontFace(GL_CW);
    }
}

void __rile_Sides(unsigned char *PC)
{
    CurAttributes->sides = *(RtFloat *)PC;
```

```
    if( CurAttributes->sides==1 ) {
      /* glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_FALSE); */
      /* glEnable(GL_CULL_FACE); */
    } else if( CurAttributes->sides==2 ) {
      /* glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE); */
      /* glDisable(GL_CULL_FACE); */
    } else {
      fprintf(stderr,"illegal risides %d\n",CurAttributes->sides);
    }
}

RtVoid __rilc_Sides(RtInt nsides)
{
    RtFloat *c;

    if( nsides!=1 && nsides!=2 ) {
      fprintf(stderr,"illegal nsides %d\n",nsides);
      return;
    }

    c = (RtFloat *)dlist_append(CurDlist,__rile_Sides,sizeof(RtFloat));
    if (c == NULL)
        return;

    c[0] = (RtFloat)nsides;

    CurAttributes->sides = nsides;
}

RtVoid __riim_Sides(RtInt nsides)
{
    if( nsides!=1 && nsides!=2 ) {
        fprintf(stderr,"illegal nsides %d\n",nsides);
        return;
    }

    CurAttributes->sides = nsides;

    if( CurAttributes->sides==1 ) {
        /* glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_FALSE); */
      /* glEnable(GL_CULL_FACE); */
    } else if( CurAttributes->sides==2 ) {
        /* glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE); */
      /* glDisable(GL_CULL_FACE); */
    }
}

RtVoid RiSides(RtInt nsides)
{
    JumpCur->Sides(nsides);
}

void __rile_Orientation(unsigned char *PC)
{
    CurAttributes->orientation = *(RtToken *)PC;
```

```
    __frontface_set();
}

RtVoid __rilc_Orientation(RtToken orientation)
{
    RtToken t, *c;

    if( !strcmp(orientation,"outside") ) {
        t = RI_OUTSIDE;
    } else if( !strcmp(orientation,"inside") ) {
        t = RI_INSIDE;
    } else if( !strcmp(orientation,"lh") ) {
        t = RI_LH;
    } else if( !strcmp(orientation,"rh") ) {
        t = RI_RH;
    } else {
        fprintf(stderr,"lc illegal orientation\n");
      return;
    }

    c = (RtToken *)dlist_append(CurDlist,__rile_Orientation,sizeof(RtToken));
    if (c == NULL)
        return;

    *c = t;

    CurAttributes->orientation = t;
}

RtVoid __riim_Orientation(RtToken orientation)
{
    if( !strcmp(orientation,"outside") ) {
        CurAttributes->orientation = RI_OUTSIDE;
    } else if( !strcmp(orientation,"inside") ) {
        CurAttributes->orientation = RI_INSIDE;
    } else if( !strcmp(orientation,"lh") ) {
        CurAttributes->orientation = RI_LH;
    } else if( !strcmp(orientation,"rh") ) {
        CurAttributes->orientation = RI_RH;
    } else {
        fprintf(stderr,"im illegal orientation\n");
        return;
    }

    __frontface_set();
}

RtVoid RiOrientation(RtToken orientation)
{
    JumpCur->Orientation(orientation);
}

/*ARGSUSED*/
void __rile_ReverseOrientation(unsigned char *PC)
{
    if( CurAttributes->orientation==RI_RH ) {
        CurAttributes->orientation = RI_LH;
```

```
    } if( CurAttributes->orientation==RI_LH ) {
        CurAttributes->orientation = RI_RH;
    } if( CurAttributes->orientation==RI_OUTSIDE ) {
        CurAttributes->orientation = RI_INSIDE;
    } if( CurAttributes->orientation==RI_INSIDE ) {
        CurAttributes->orientation = RI_OUTSIDE;
    }
    __frontface_set();
}

RtVoid __rilc_ReverseOrientation(void)
{
    dlist_append(CurDlist,__rile_ReverseOrientation,0);
}

RtVoid __riim_ReverseOrientation(void)
{
    if( CurAttributes->orientation==RI_RH ) {
        CurAttributes->orientation = RI_LH;
    } if( CurAttributes->orientation==RI_LH ) {
        CurAttributes->orientation = RI_RH;
    } if( CurAttributes->orientation==RI_OUTSIDE ) {
        CurAttributes->orientation = RI_INSIDE;
    } if( CurAttributes->orientation==RI_INSIDE ) {
        CurAttributes->orientation = RI_OUTSIDE;
    }
    __frontface_set();
}

RtVoid RiReverseOrientation(void)
{
    JumpCur->ReverseOrientation();
}


/*ARGSUSED*/
void __rile_Identity(unsigned char *PC)
{
    glLoadIdentity();
}

RtVoid __rilc_Identity(void)
{
    dlist_append(CurDlist,__rile_Identity,0);
    glLoadIdentity();
}

RtVoid __riim_Identity(void)
{
    glLoadIdentity();
}

RtVoid RiIdentity(void)
{
    JumpCur->Identity();
}
```

```
void __rile_Transform(unsigned char *PC)
{
    RtMatrix *m = (RtMatrix *)PC;

#if 0
        if( CurAttributes->transorient==RI_RH ) {
            CurAttributes->transorient = RI_LH;
        } else {
            CurAttributes->transorient = RI_RH;
        }
#endif

    glLoadMatrixf((GLfloat *)(*m));
}

RtVoid __rilc_Transform(RtMatrix transform)
{
    RtMatrix *m;

    m = (RtMatrix *)dlist_append(CurDlist,__rile_Transform,sizeof(RtMatrix));
    if (m == NULL)
        return;

    bcopy(transform,m,sizeof(RtMatrix));

    glLoadMatrixf((GLfloat *)transform);
}

RtVoid __riim_Transform(RtMatrix transform)
{
#if 0
    /* XXX hard-coded for the bikeshop and red */
    if( CurAttributes->transorient==RI_RH ) {
        CurAttributes->transorient = RI_LH;
    } else {
        CurAttributes->transorient = RI_RH;
    }
#endif

    glLoadMatrixf((GLfloat *)transform);
}

RtVoid RiTransform(RtMatrix transform)
{
    JumpCur->Transform(transform);
}

void __rile_ConcatTransform(unsigned char *PC)
{
    RtMatrix *m = (RtMatrix *)PC;

    glMultMatrixf((GLfloat *)(*m));
}

RtVoid __rilc_ConcatTransform(RtMatrix transform)
{
    RtMatrix *m;
```

```
    m = (RtMatrix
*)dlist_append(CurDlist,__rile_ConcatTransform,sizeof(RtMatrix));
    if (m == NULL)
        return;

    bcopy(transform,m,sizeof(RtMatrix));

    glMultMatrixf((GLfloat *)transform);
}

RtVoid __riim_ConcatTransform(RtMatrix transform)
{
    glMultMatrixf((GLfloat *)transform);
}

RtVoid RiConcatTransform(RtMatrix transform)
{
    JumpCur->ConcatTransform(transform);
}

void __rile_Perspective(unsigned char *PC)
{
    RtFloat fov = *(RtFloat *)PC;

    gluPerspective((GLdouble)fov,1.,1.,1000000.);
}

RtVoid __rilc_Perspective(RtFloat fov)
{
    RtFloat *c;

    c = (RtFloat *)dlist_append(CurDlist,__rile_Perspective,sizeof(RtFloat));
    if (c == NULL)
        return;

    *c = fov;

    gluPerspective((GLdouble)fov,1.,1.,1000000.);
}

RtVoid __riim_Perspective(RtFloat fov)
{
    gluPerspective((GLdouble)fov,1.,1.,1000000.);
}

RtVoid RiPerspective(RtFloat fov)
{
    JumpCur->Perspective(fov);
}

void __rile_Translate(unsigned char *PC)
{
    RtFloat *d = (RtFloat *)PC;

    glTranslatef(d[0],d[1],d[2]);
}
```

```
RtVoid __rilc_Translate(RtFloat dx, RtFloat dy, RtFloat dz)
{
    RtFloat *d;

    d = (RtFloat *)dlist_append(CurDlist,__rile_Translate,3*sizeof(RtFloat));
    if (d == NULL)
        return;

    d[0] = dx;
    d[1] = dy;
    d[2] = dz;

    glTranslatef(dx,dy,dz);
}

RtVoid __riim_Translate(RtFloat dx, RtFloat dy, RtFloat dz)
{
    glTranslatef(dx,dy,dz);
}

RtVoid RiTranslate(RtFloat dx, RtFloat dy, RtFloat dz)
{
    JumpCur->Translate(dx,dy,dz);
}

void __rile_Rotate(unsigned char *PC)
{
    RtFloat *d = (RtFloat *)PC;

    glRotatef(d[0],d[1],d[2],d[3]);
}

RtVoid __rilc_Rotate(RtFloat angle, RtFloat dx, RtFloat dy, RtFloat dz)
{
    RtFloat *d;

    d = (RtFloat *)dlist_append(CurDlist,__rile_Rotate,4*sizeof(RtFloat));
    if (d == NULL)
        return;

    d[0] = angle;
    d[1] = dx;
    d[2] = dy;
    d[3] = dz;

    glRotatef(angle,dx,dy,dz);
}

RtVoid __riim_Rotate(RtFloat angle, RtFloat dx, RtFloat dy, RtFloat dz)
{
    glRotatef(angle,dx,dy,dz);
}

RtVoid RiRotate(RtFloat angle, RtFloat dx, RtFloat dy, RtFloat dz)
{
    JumpCur->Rotate(angle,dx,dy,dz);
```

```
}

void __rile_Scale(unsigned char *PC)
{
    RtFloat *d = (RtFloat *)PC;

    if( d[0]*d[1]*d[2]<0. ) {
      if( CurAttributes->transorient==RI_RH ) {
          CurAttributes->transorient = RI_LH;
      } else {
          CurAttributes->transorient = RI_RH;
      }
      __frontface_set();
    }

    glScalef(d[0],d[1],d[2]);
}

RtVoid __rilc_Scale(RtFloat dx, RtFloat dy, RtFloat dz)
{
    RtFloat *d;

    d = (RtFloat *)dlist_append(CurDlist,__rile_Scale,3*sizeof(RtFloat));
    if (d == NULL)
        return;

    d[0] = dx;
    d[1] = dy;
    d[2] = dz;

    if( d[0]*d[1]*d[2]<0. ) {
      if( CurAttributes->transorient==RI_RH ) {
          CurAttributes->transorient = RI_LH;
      } else {
          CurAttributes->transorient = RI_RH;
      }
    }

    glScalef(dx,dy,dz);
}

RtVoid __riim_Scale(RtFloat dx, RtFloat dy, RtFloat dz)
{
    if( dx*dy*dz<0. ) {
      if( CurAttributes->transorient==RI_RH ) {
          CurAttributes->transorient = RI_LH;
      } else {
          CurAttributes->transorient = RI_RH;
      }
      __frontface_set();
    }

    glScalef(dx,dy,dz);
}

/*ARGSUSED*/
RtVoid RiScale(RtFloat dx, RtFloat dy, RtFloat dz)
```

```
{
    JumpCur->Scale(dx,dy,dz);
}

/*ARGSUSED*/
RtVoid RiSkew(RtFloat angle, RtFloat dx1, RtFloat dy1, RtFloat dz1,
            RtFloat dx2, RtFloat dy2, RtFloat dz2)
{
    fprintf(stderr,"RiSkew unimplemented\n");
}

/*ARGSUSED*/
RtVoid RiDeformationV(char *name, RtInt n, RtToken tokens[], RtPointer
params[])
{
    fprintf(stderr,"RiDeformation unimplemented\n");
}

/*ARGSUSED*/
RtVoid RiDeformation(char *name, ...)
{
    fprintf(stderr,"RiDeformation unimplemented\n");
}

/*ARGSUSED*/
RtVoid RiDisplacementV(char *name, RtInt n, RtToken tokens[], RtPointer
params[])
{
    /* fprintf(stderr,"RiDisplacement unimplemented\n"); */
}

/*ARGSUSED*/
RtVoid RiDisplacement(char *name, ...)
{
    /* fprintf(stderr,"RiDisplacement unimplemented\n"); */
}

#define ORIENTSTACKDEPTH 256
static RtToken orientstack[ORIENTSTACKDEPTH];
static int norient = 0;

/*ARGSUSED*/
void __rile_TransformBegin(unsigned char *PC)
{
    glPushMatrix();
    orientstack[norient++] = CurAttributes->transorient;
}

RtVoid __rilc_TransformBegin(void)
{
    dlist_append(CurDlist,__rile_TransformBegin,0);

    glPushMatrix();
    orientstack[norient++] = CurAttributes->transorient;
}

RtVoid __riim_TransformBegin(void)
```

```
{
    glPushMatrix();
    orientstack[norient++] = CurAttributes->transorient;
}

RtVoid RiTransformBegin(void)
{
    JumpCur->TransformBegin();
}

/*ARGSUSED*/
void __rile_TransformEnd(unsigned char *PC)
{
    glPopMatrix();
    CurAttributes->transorient = orientstack[--norient];

    __frontface_set();
}

RtVoid __rilc_TransformEnd(void)
{
    dlist_append(CurDlist,__rile_TransformEnd,0);

    glPopMatrix();
    CurAttributes->transorient = orientstack[--norient];
}

void __riim_TransformEnd(void)
{
    glPopMatrix();
    CurAttributes->transorient = orientstack[--norient];

    __frontface_set();
}

RtVoid RiTransformEnd(void)
{
    JumpCur->TransformEnd();
}

/* linked list of coordinate systems */
static CoordSystem *ricoord = NULL;
extern void invert_matrix(RtMatrix a, RtMatrix m);

RtVoid RiCoordinateSystem(RtToken space)
{
    CoordSystem *c;

    c = (CoordSystem *)malloc(sizeof(CoordSystem));
    strcpy(c->name,space);
    glGetFloatv(GL_MODELVIEW_MATRIX,(GLfloat *)c->xform);
    invert_matrix(c->xform,c->ixform);

    c->next = ricoord;
    ricoord = c;
}
```

```
static CoordSystem *lookup_coordsystem(RtToken name)
{
    CoordSystem *c = ricoord;

    while( c!=NULL ) {
      if( strcmp(c->name,name)==0 )
          return( c );
      c = c->next;
    }
    return(NULL);
}

RtPoint *RiTransformPoints(RtToken fromspace, RtToken tospace,
                                    RtInt npoints, RtPoint *points)
{
    int i;
    CoordSystem *f = lookup_coordsystem(fromspace);
    CoordSystem *t = lookup_coordsystem(tospace);
    extern void transform_point(RtFloat *p, RtFloat *result, RtMatrix xform);
    RtFloat p[4];

    /* wc = fromspace . pointf = tospace . pointt
       pointt = tospaceinv . fromspace . pointf */

    for( i=0; i<npoints; i++ ) {
        transform_point(points[i], p, f->xform);
        transform_point(p, points[i], t->ixform);
    }

    return points;
}
```